2

REPORT D ... ATION PAGE

Form Approved
OPM No. 0704-0188

AD-A236 326

1. AGE... ...ATE	3. REPORT TYPE AND DATES COVERED Final: 28 Nov 1991 to 03 Mar 1993
4. TITLE AND SUBTITLE TeleSoft, IBM Ada 370, Version 1.1.0, IBM 3083 (Host & Target), 901128W1.11091	5. FUNDING NUMBERS
6. AUTHOR(S) Wright-Patterson AFB, Dayton, OH USA	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Ada Validation Facility, Language Control Facility ASD/SCEL Bldg. 676, Rm 135 Wright-Patterson AFB Dayton, OH 45433	8. PERFORMING ORGANIZATION REPORT NUMBER AVF_VSR_422.0491
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Ada Joint Program Office United States Department of Defense Pentagon, Rm 3E114 Washington, D.C. 20301-3081	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

TeleSoft, IBM Ada 370, Version 1.1.0, Wright-Patterson AFB, IBM 3083 under VM/SP HPO, Release 5.0 (Host & Target), ACVC 1.11.

14. SUBJECT TERMS Ada programming language, Ada Compiler Val. Summary Report, Ada Compiler Val. Capability, Val. Testing, Ada Val. Office, Ada Val. Facility, ANSI/MIL-STD-1815A, AJPO.			15. NUMBER OF PAGES 16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION UNCLASSIFED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT

Certificate Information

The following Ada implementation was tested and determined to pass ACVC 1.11. Testing was completed on 28 November 1991.

Compiler Name and Version: IBM Ada/370, Version 1.1.0

Host Computer System: IBM 3083 under VM/SP HPO, Release 5.0

Target Computer System: IBM 3083 under VM/SP HPO, Release 5.0

See Section 3.1 for any additional information about the testing environment.

As a result of this validation effort, Validation Certificate 901128W1.11091 is awarded to TeleSoft. This certificate expires on 1 March 1993.

This report has been reviewed and is approved.

[illegible]

Ada Validation Facility
Steven P. Wilson
Technical Director
ASD/SCEL
Wright-Patterson AFB OH 45433-6503

[illegible]

Ada Validation Organization
Director, Computer & Software Engineering Division
Institute for Defense Analyses
Alexandria VA 22311

[illegible]

Ada Joint Program Office
Dr. John Solomond, Director
Department of Defense
Washington DC 20301

[illegible]

A-1

QUALITY INSPECTED

91-00490

91 5 24 012

AVF Control Number: AVF-VSR-422.0491
18 April 1991
90-06-28-TEL

Ada COMPILER VALIDATION SUMMARY REPORT:

Certificate Number: 901128W1.11091
TeleSoft
IBM Ada 370, Version 1.1.0
IBM 3083 => IBM 3083

Prepared By:
Ada Validation Facility
ASD/SCEL
Wright Patterson AFB OH 45433-6503

Certificate Information

The following Ada implementation was tested and determined to pass ACVC 1.11. Testing was completed on 28 November 1991.

Compiler Name and Version:	IBM Ada/370, Version 1.1.0
Host Computer System:	IBM 3083 under VM/SP HPO, Release 5.0
Target Computer System:	IBM 3083 under VM/SP HPO, Release 5.0

See Section 3.1 for any additional information about the testing environment.

As a result of this validation effort, Validation Certificate 901128W1.11091 is awarded to TeleSoft. This certificate expires on 1 March 1993.

This report has been reviewed and is approved.

Steven P. Wilson

Ada Validation Facility
Steven P. Wilson
Technical Director
ASD/SCEL
Wright-Patterson AFB OH 45433-6503

[illegible]

Ada Validation Organization
Director, Computer & Software Engineering Division
Institute for Defense Analyses
Alexandria VA 22311

Ada Joint Program Office
Dr. John Solomond, Director
Department of Defense
Washington DC 20301

DECLARATION OF CONFORMANCE

Compiler Implementor: TeleSoft
Ada Validation Facility: ASD/SCEL, Wright-Patterson AFB, OH 45433-6503
Ada Compiler Validation Capability (ACVC) Version: 1.11

Base Configuration

Base Compiler Name: IBM Ada 370, Version 1.1.0
Host Architecture ISA: IBM 3083
Operating System: VM/SP HPO Release 5.0

Target Architecture ISA: IBM 3083
Operating System: VM/SP HPO Release 5.0

Implementor's Declaration

I, the undersigned, representing TeleSoft have implemented no deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the compiler listed in this declaration. I declare that International Business Machines Corporation is the owner of record of the object code of the Ada language compiler listed above and, as such, is responsible for maintaining said compiler in conformance to ANSI/MIL-STD-1815A. All certificates and registrations for the Ada language compiler listed in this declaration shall be made only in the owner's corporate name.

[illegible]
TeleSoft
Raymond A. Parra, Director, Contracts & Legal

Date: November 28, 1990

Owner's Declaration

I, the undersigned, representing International Business Machines Corporation take full responsibility for implementation and maintenance of the Ada compiler listed above, and agree to the public disclosure of the final Validation Summary Report. I declare that all of the Ada language compilers listed, and their host/target performance are in compliance with the Ada Language Standard ANSI/MIL-STD-1815A.

[illegible]
International Business Machines Corporation
Yim Chan, Ada Development Manager

Date: Nov 28, 1990

CONTENTS

CHAPTER 1

INTRODUCTION

The Ada implementation described above was tested according to the Ada Validation Procedures [Pro90] against the Ada Standard [Ada83] using the current Ada Compiler Validation Capability (ACVC). This Validation Summary Report (VSR) gives an account of the testing of this Ada implementation. For any technical terms used in this report, the reader is referred to [Pro90]. A detailed description of the ACVC may be found in the current ACVC User's Guide [UG89].

1.1 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the Ada Certification Body may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject implementation has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from the AVF which performed this validation or from:

National Technical Information Service
5285 Port Royal Road
Springfield VA 22161

Questions regarding this report or the validation test results should be directed to the AVF which performed this validation or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

1.2 REFERENCES

[Ada83] Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

[Pro90] Ada Compiler Validation Procedures, Version 2.1, Ada Joint Program Office, August 1990.

[UG89] Ada Compiler Validation Capability User's Guide, 21 June 1989.

1.3 ACVC TEST CLASSES

Compliance of Ada implementations is tested by means of the ACVC. The ACVC contains a collection of test programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable. Class B and class L tests are expected to produce errors at compile time and link time, respectively.

The executable tests are written in a self-checking manner and produce a PASSED, FAILED, or NOT APPLICABLE message indicating the result when they are executed. Three Ada library units, the packages REPORT and SPPRT13, and the procedure CHECK_FILE are used for this purpose. The package REPORT also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The package SPPRT13 is used by many tests for Chapter 13 of the Ada Standard. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for Chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. If these units are not operating correctly, validation testing is discontinued.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that all violations of the Ada Standard are detected. Some of the class B tests contain legal Ada code which must not be flagged illegal by the compiler. This behavior is also verified.

Class L tests check that an Ada implementation correctly detects violation of the Ada Standard involving multiple, separately compiled units. Errors are expected at link time, and execution is attempted.

In some tests of the ACVC, certain macro strings have to be replaced by implementation-specific values -- for example, the largest integer. A list of the values used for this implementation is provided in Appendix A. In addition to these anticipated test modifications, additional changes may be required to remove unforeseen conflicts between the tests and implementation-dependent characteristics. The modifications required for this implementation are described in section 2.3.

For each Ada implementation, a customized test suite is produced by the AVF. This customization consists of making the modifications described in the preceding paragraph, removing withdrawn tests (see section 2.1) and, possibly some inapplicable tests (see Section 2.2 and [UG89]).

In order to pass an ACVC an Ada implementation must process each test of the customized test suite according to the Ada Standard.

1.4 DEFINITION OF TERMS

Ada Compiler	The software and any needed hardware that have to be added to a given host and target computer system to allow transformation of Ada programs into executable form and execution thereof.
Ada Compiler Validation Capability (ACVC)	The means for testing compliance of Ada implementations, consisting of the test suite, the support programs, the ACVC user's guide and the template for the validation summary report.
Ada Implementation	An Ada compiler with its host computer system and its target computer system.
Ada Validation Facility (AVF)	The part of the certification body which carries out the procedures required to establish the compliance of an Ada implementation.
Ada Validation Organization (AVO)	The part of the certification body that provides technical guidance for operations of the Ada certification system.
Compliance of an Ada Implementation	The ability of the implementation to pass an ACVC version.
Computer System	A functional unit, consisting of one or more computers and associated software, that uses common storage for all or part of a program and also for all or part of the data necessary for the execution of the program; executes user-written or user-designated programs; performs user-designated data manipulation, including arithmetic operations and logic operations; and that can execute programs that modify themselves during execution. A computer system may be a stand-alone unit or may consist of several inter-connected units.
Conformity	Fulfillment by a product, process or service of all requirements specified.

Customer	An individual or corporate entity who enters into an agreement with an AVF which specifies the terms and conditions for AVF services (of any kind) to be performed.
Declaration of Conformance	A formal statement from a customer assuring that conformity is realized or attainable on the Ada implementation for which validation status is realized.
Host Computer System	A computer system where Ada source programs are transformed into executable form.
Inapplicable test	A test that contains one or more test objectives found to be irrelevant for the given Ada implementation.
Operating System	Software that controls the execution of programs and that provides services such as resource allocation, scheduling, input/output control, and data management. Usually, operating systems are predominantly software, but partial or complete hardware implementations are possible.
Target Computer System	A computer system where the executable form of Ada programs are executed.
Validated Ada Compiler	The compiler of a validated Ada implementation.
Validated Ada Implementation	An Ada implementation that has been validated successfully either by AVF testing or by registration [Pro90].
Validation	The process of checking the conformity of an Ada compiler to the Ada programming language and of issuing a certificate for this implementation.
Withdrawn test	A test found to be incorrect and not used in conformity testing. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains erroneous or illegal use of the Ada programming language.

CHAPTER 2

IMPLEMENTATION DEPENDENCIES

2.1 WITHDRAWN TESTS

The following tests have been withdrawn by the AVO. The rationale for withdrawing each test is available from either the AVO or the AVF. The publication date for this list of withdrawn tests is 21 November 1990.

E28005C	B28006C	C34006D	C35702A	B41308B	C43004A
C45114A	C45346A	C45612B	C45651A	C46022A	B49008A
A74006A	C74308A	B83022B	B83022H	B83025B	B83025D
B83026B	B85001L	C83026A	C83041A	C97116A	C98003B
BA2011A	CB7001A	CB7001B	CB7004A	CC1223A	BC1226A
CC1226B	BC3009B	BD1B02B	BD1B06A	AD1B08A	BD2A02A
CD2A21E	CD2A23E	CD2A32A	CD2A41A	CD2A41E	CD2A87A
CD2B15C	BD3006A	BD4008A	CD4022A	CD4022D	CD4024B
CD4024C	CD4024D	CD4031A	CD4051D	CD5111A	CD7004C
ED7005D	CD7005E	AD7006A	CD7006E	AD7201A	AD7201E
CD7204B	BD8002A	BD8004C	CD9005A	CD9005B	CDA201E
CE2107I	CE2117A	CE2117B	CE2119B	CE2205B	CE2405A
CE3111C	CE3116A	CE3118A	CE3411B	CE3412B	CE3607B
CE3607C	CE3607D	CE3812A	CE3814A	CE3902B	

2.2 INAPPLICABLE TESTS

A test is inapplicable if it contains test objectives which are irrelevant for a given Ada implementation. Reasons for a test's inapplicability may be supported by documents issued by ISO and the AJPO known as Ada Commentaries and commonly referenced in the format AI-ddddd. For this implementation, the following tests were determined to be inapplicable for the reasons indicated; references to Ada Commentaries are included as appropriate.

The following 201 tests have floating-point type declarations requiring more digits than SYSTEM.MAX_DIGITS:

C24113L..Y (14 tests)	C35705L..Y (14 tests)
C35706L..Y (14 tests)	C35707L..Y (14 tests)
C35708L..Y (14 tests)	C35802L..Z (15 tests)
C45241L..Y (14 tests)	C45321L..Y (14 tests)
C45421L..Y (14 tests)	C45521L..Z (15 tests)
C45524L..Z (15 tests)	C45621L..Z (15 tests)
C45641L..Y (14 tests)	C46012L..Z (15 tests)

The following 21 tests check for the predefined type LONG_INTEGER:

C35404C	C45231C	C45304C	C45411C	C45412C
C45502C	C45503C	C45504C	C45504F	C45611C
C45612C	C45613C	C45614C	C45631C	C45632C
B52004D	C55B07A	B55B09C	B86001W	C86006C
CD7101F				

C35404D, C45231D, B86001X, C86006E, and CD7101G check for a predefined integer type with a name other than INTEGER, LONG_INTEGER, or SHORT_INTEGER.

C35508I..J and C35508M..N (4 tests) include enumeration representation clauses for boolean types in which the specified values are other than (FALSE => 0, TRUE => 1); this implementation does not support a change in representation for boolean types. (See section 2.3.)

C35713B, C45423B, B86001T, and C86006H check for the predefined type SHORT_FLOAT.

C35713D and B86001Z check for a predefined floating-point type with a name other than FLOAT, LONG_FLOAT, or SHORT_FLOAT.

C45423A, C45523A, and C45622A check that if MACHINE_OVERFLOWS is TRUE and the results of various floating-point operations lie outside the range of the base type, then the proper exception is raised. For this implementation, MACHINE_OVERFLOWS is FALSE.

C45531M..P (4 tests) and C45532M..P (4 tests) check fixed-point operations for types that require a SYSTEM.MAX_MANTISSA of 47 or greater; for this implementation, MAX_MANTISSA is less than 47.

C86001F recompiles package SYSTEM, making package TEXT_IO, and hence package REPORT, obsolete. For this implementation, the package TEXT_IO is dependent upon package SYSTEM.

B86001Y checks for a predefined fixed-point type other than DURATION.

C96005B checks for values of type DURATION'BASE that are outside the range of DURATION. There are no such values for this implementation.

CA2009C, CA2009F, BC3204C, and BC3205D instantiate generic units before their bodies are compiled; this implementation creates a dependence on generic units as allowed by AI-00408 and AI-00530 such that the compilation of the generic unit bodies makes the instantiating units obsolete.

LA3004A, LA3004B, EA3004C, EA3004D, CA3004E, and CA3004F check for pragma INLINE for procedures and functions.

CD1009C uses a representation clause specifying a non-default size for a floating-point type.

CD2A84A, CD2A84E, CD2A84I..J (2 tests), and CD2A84O use representation clauses specifying non-default sizes for access types.

BD8001A, BD8003A, BD8004A..B (2 tests), and AD8011A use machine code insertions.

The tests listed in the following table are not applicable because the given file operations are supported for the given combination of mode and file access method:

Test	File Operation	Mode	File Access Method
CE2102D	CREATE	IN_FILE	SEQUENTIAL_IO
CE2102E	CREATE	OUT_FILE	SEQUENTIAL_IO
CE2102F	CREATE	INOUT_FILE	DIRECT_IO
CE2102I	CREATE	IN_FILE	DIRECT_IO
CE2102J	CREATE	OUT_FILE	DIRECT_IO
CE2102N	OPEN	IN_FILE	SEQUENTIAL_IO
CE2102O	RESET	IN_FILE	SEQUENTIAL_IO
CE2102P	OPEN	OUT_FILE	SEQUENTIAL_IO
CE2102Q	RESET	OUT_FILE	SEQUENTIAL_IO
CE2102R	OPEN	INOUT_FILE	DIRECT_IO
CE2102S	RESET	INOUT_FILE	DIRECT_IO
CE2102T	OPEN	IN_FILE	DIRECT_IO
CE2102U	RESET	IN_FILE	DIRECT_IO
CE2102V	OPEN	OUT_FILE	DIRECT_IO
CE2102W	RESET	OUT_FILE	DIRECT_IO
CE3102E	CREATE	IN_FILE	TEXT_IO
CE3102F	RESET	Any Mode	TEXT_IO
CE3102G	DELETE	---------	TEXT_IO
CE3102I	CREATE	OUT_FILE	TEXT_IO
CE3102J	OPEN	IN_FILE	TEXT_IO
CE3102K	OPEN	OUT_FILE	TEXT_IO

AE2101H, EE2401D, and EE2401G use instantiations of package DIRECT_IO with unconstrained array types and record types with discriminants without defaults. These instantiations are rejected by this compiler.

The following 16 tests check operations on sequential, direct, and text files when multiple internal files are associated with the same external file and one or more are open for writing; USE_ERROR is raised when this association is attempted.

CE2107B..E	CE2107G..H	CE2107L	CD2110B	CE2110D
CE2111D	CE2111H	CE3111B	CE3111D..E	CE3114B
CE3115A				

CE2203A checks that WRITE raises USE_ERROR if the capacity of the external file is exceeded for SEQUENTIAL_IO. This implementation does not restrict file capacity.

CE2403A checks that WRITE raises USE_ERROR if the capacity of the external file is exceeded for DIRECT_IO. This implementation does not restrict file capacity.

CE3413B checks that PAGE raises LAYOUT_ERROR when the value of the page number exceeds COUNT'LAST. For this implementation, the value of COUNT'LAST is greater than 150000 making the checking of this objective impractical.

2.3 TEST MODIFICATIONS

Modifications (see section 1.3) were required for 31 tests:

The following tests were split into two or more tests because this implementation did not report the violations of the Ada Standard in the way expected by the original tests.

BA1001A	BA2001C	BA2001E	BA3006A	BA3006B
BA3007B	BA3008A	BA3008B	BA3013A	

C35508I..J and C35508M..N (4 tests) were graded inapplicable by Evaluation Modification as directed by the AVO. These tests attempt to change the representation of a boolean type. The AVO ruled that, in consideration of the particular nature of boolean types and the operations that are defined for the type and for arrays of the type, a change of representation need not be supported; the ARG will address this issue in Commentary AI-00564.

C52008B was graded passed by Test Modifications as directed by the AVO. This test uses a record type with discriminants with defaults and that has array components whose size depends on the values of some discriminants of type INTEGER. On compilation of the type declaration, this implementation raises NUMERIC_ERROR as it attempts to calculate the maximum possible size for objects of the type. Although this behavior is a violation of the Ada standard, the AVO ruled that the implementation be accepted for validation in consideration of intended changes to the standard to allow for compile-time detection of run-time error conditions. The test was modified to constrain the subtype of the discriminants. Line 16 was modified to declare a constrained subtype of INTEGER, and discriminant declarations in lines 17 and 25 were modified to use that subtype; the lines are given below:

```
16   SUBTYPE SUBINT IS INTEGER RANGE -128 .. 127;
17   TYPE REC1(D1,D2 : SUBINT) IS

25   TYPE REC2(D1,D2,D3,D4 : SUBINT := 0) IS
```

CD1009A, CD1009I, CD1C03A, CD2A21C, CD2A24A, and CD2A31A..C (3 tests) were graded passed by Evaluation Modification as directed by the AVO. These tests use instantiations of the support procedure Length_Check, which uses Unchecked_Conversion according to the interpretation given in AI-00590. The AVO ruled that this interpretation is not binding under ACVC 1.11; the tests are ruled to be passed if they produce Failed messages only from the instances of Length_Check--i.e, the allowed Report.Failed messages have the general form:

```
" * CHECK ON REPRESENTATION FOR <TYPE_ID> FAILED."
```

CE2103C..D (2 tests) were graded passed by Test Modification as directed by the AVO. These tests close an empty file; however, the IBM VM/SP HPO operating system does not allow an empty file to exist, so the file is deleted and USE_ERROR is raised. The AVO ruled that this behavior is acceptable, given the operating system (cf. AI-00325); the AVO directed that the tests be modified and passed with the following write statement inserted into the two tests, respectively, at lines 56 and 55:

```
WRITE (TEST_FILE_ONE, 'A');
```

EE3301B, EE3405B, and EE3410F were graded passed by Evaluation Modification as directed by the AVO. These tests check certain I/O operations on the current default output file, including standard output. This implementation outputs the ASCII form-feed character which has no effect on the standard IBM output devices; in general, there is no common form-feed mechanism for the devices. Thus, the printed output from this test did not contain the expected page breaks. The AVO ruled that these tests should be considered passed if none of the tests' internal checks was failed (i.e., if the tests report "TENTATIVELY PASSED").

CHAPTER 3

PROCESSING INFORMATION

3.1 TESTING ENVIRONMENT

The Ada implementation tested in this validation effort is described adequately by the information given in the initial pages of this report.

For a point of contact for technical information about this Ada implementation system, see:

IBM Canada, Ltd
844 Don Mills Road
North York, Ontario
Canada M3C IB7
ATTN: Antony Niro
31/257/844/TOR

For a point of contact for sales information about this Ada implementation system, see:

IBM Canada, Ltd
844 Don Mills Road
North York, Ontario
Canada M3C IB7
ATTN: Yim Chan
31/257/844/TOR

Testing of this Ada implementation was conducted at the customer's site by a validation team from the AVF.

3.2 SUMMARY OF TEST RESULTS

An Ada Implementation passes a given ACVC version if it processes each test of the customized test suite in accordance with the Ada Programming Language Standard, whether the test is applicable or inapplicable; otherwise, the Ada Implementation fails the ACVC [Pro90].

PROCESSING INFORMATION

For all processed tests (inapplicable and applicable), a result was obtained that conforms to the Ada Programming Language Standard.

a) Total Number of Applicable Tests	3772
b) Total Number of Withdrawn Tests	83
c) Processed Inapplicable Tests	114
d) Non-Processed I/O Tests	0
e) Non-Processed Floating-Point Precision Tests	201
f) Total Number of Inapplicable Tests	315
g) Total Number of Tests for ACVC 1.11	4170

All I/O tests of the test suite were processed because this implementation supports a file system. The above number of floating-point tests were not processed because they used floating-point precision exceeding that supported by the implementation. When this compiler was tested, the tests listed in section 2.1 had been withdrawn because of test errors.

3.3 TEST EXECUTION

Version 1.11 of the ACVC comprises 4170 tests. When this compiler was tested, the tests listed in section 2.1 had been withdrawn because of test errors. The AVF determined that 315 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 201 executable tests that use floating-point precision exceeding that supported by the implementation. In addition, the modified tests mentioned in section 2.3 were also processed.

A magnetic tape containing the customized test suite (see section 1.3) was taken on-site by the validation team for processing. The contents of the tape were loaded directly onto the host computer.

After the test files were loaded onto the host computer, the full set of tests was processed by the Ada implementation.

Testing was performed using command scripts provided by the customer and reviewed by the validation team. See Appendix B for a complete listing of the processing options for this implementation. It also indicates the default options. The options invoked explicitly for validation testing during this test were:

1. CLEAN
2. ERROR(LIST)
3. LIST(ERR1)
4. RUN(TEXT)

Test output, compiler and linker listings, and job logs were captured on magnetic tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

APPENDIX A

MACRO PARAMETERS

This appendix contains the macro parameters used for customizing the ACVC. The meaning and purpose of these parameters are explained in [UG89]. The parameter values are presented in two tables. The first table lists the values that are defined in terms of the maximum input-line length, which is the value for $MAX_IN_LEN--also listed here. These values are expressed here as Ada string aggregates, where "V" represents the maximum input-line length.

Macro Parameter	Macro Value
$BIG_ID1	(1..V-1 => 'A', V => '1')
$BIG_ID2	(1..V-1 => 'A', V => '2')
$BIG_ID3	(1..V/2 => 'A') & '3' & (1..V-1-V/2 => 'A')
$BIG_ID4	(1..V/2 => 'A') & '4' & (1..V-1-V/2 => 'A')
$BIG_INT_LIT	(1..V-3 => '0') & "298"
$BIG_REAL_LIT	(1..V-5 => '0') & "690.0"
$BIG_STRING1	'"' & (1..V/2 => 'A') & '"'
$BIG_STRING2	'"' & (1..V-1-V/2 => 'A') & '1' & '"'
$BLANKS	(1..V-20 => ' ')
$MAX_LEN_INT_BASED_LITERAL	"2:" & (1..V-5 => '0') & "11:"
$MAX_LEN_REAL_BASED_LITERAL	"16:" & (1..V-7 => '0') & "F.E:"
$MAX_STRING_LITERAL	'"' & (1..V-2 => 'A') & '"'

MACRO PARAMETERS

The following table lists all of the other macro parameters and their respective values:

Macro Parameter	Macro Value
$MAX_IN_LEN	200
$ACC_SIZE	32
$ALIGNMENT	4
$COUNT_LAST	2_147_483_646
$DEFAULT_MEM_SIZE	16777215
$DEFAULT_STOR_UNIT	8
$DEFAULT_SYS_NAME	IBM370
$DELTA_DOC	2#1.0#E-31
$ENTRY_ADDRESS	ENT_ADDRESS
$ENTRY_ADDRESS1	ENT_ADDRESS1
$ENTRY_ADDRESS2	ENT_ADDRESS2
$FIELD_LAST	1000
$FILE_TERMINATOR	' '
$FIXED_NAME	NO_SUCH_TYPE
$FLOAT_NAME	NO_SUCH_TYPE
$FORM_STRING	' '
$FORM_STRING2	"CANNOT_RESTRICT_FILE_CAPACITY"
$GREATER_THAN_DURATION	86401.0
$GREATER_THAN_DURATION_BASE_LAST	131073.0
$GREATER_THAN_FLOAT_BASE_LAST	7.2370052E+75
$GREATER_THAN_FLOAT_SAFE_LARGE	7.237004E+75

$GREATER_THAN_SHORT_FLOAT_SAFE_LARGE	7.237E+75
$HIGH_PRIORITY	255
$ILLEGAL_EXTERNAL_FILE_NAME1	"BADCHAR*%"
$ILLEGAL_EXTERNAL_FILE_NAME2	"BAD-CHARS!@~"
$INAPPROPRIATE_LINE_LENGTH	1029
$INAPPROPRIATE_PAGE_LENGTH	-1
$INCLUDE_PRAGMA1	'PRAGMA INCLUDE ("A28006D1.TST");'
$INCLUDE_PRAGMA2	'PRAGMA INCLUDE ("B28006F1.TST");'
$INTEGER_FIRST	-2147483648
$INTEGER_LAST	2147483647
$INTEGER_LAST_PLUS_1	2147483648
$INTERFACE_LANGUAGE	C
$LESS_THAN_DURATION	-86401.0
$LESS_THAN_DURATION_BASE_FIRST	131073.0
$LINE_TERMINATOR	' '
$LOW_PRIORITY	0
$MACHINE_CODE_STATEMENT	NULL;
$MACHINE_CODE_TYPE	NO_SUCH_TYPE
$MANTISSA_DOC	31
$MAX_DIGITS	15
$MAX_INT	2147483647
$MAX_INT_PLUS_1	2147483647
$MIN_INT	-2147483648

MACRO PARAMETERS

$NAME	NO_SUCH_TYPE_AVAILABLE
$NAME_LIST	mc68000,anuyk44,ibm370
$NAME_SPECIFICATION1	"X2120A DATA A1"
$NAME_SPECIFICATION2	"X2120B DATA A1"
$NAME_SPECIFICATION3	"X3119A DATA A1"
$NEG_BASED_INT	16#FFFFFFFE#
$NEW_MEM_SIZE	16777215
$NEW_STOR_UNIT	8
$NEW_SYS_NAME	IBM370
$PAGE_TERMINATOR	' '
$RECORD_DEFINITION	"NEW INTEGER;"
$RECORD_NAME	NO_SUCH_MACHINE_CODE_TYPE
$TASK_SIZE	32
$TASK_STORAGE_SIZE	1024
$TICK	0.000001
$VARIABLE_ADDRESS	VAR_ADDRESS
$VARIABLE_ADDRESS1	VAR_ADDRESS1
$VARIABLE_ADDRESS2	VAR_ADDRESS2
$YOUR_PRAGMA	PRIORITY

APPENDIX B

COMPILATION SYSTEM OPTIONS

The compiler options of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this Appendix are to compiler documentation and not to this report.

LINKER OPTIONS

The linker options of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this Appendix are to linker documentation and not to this report.

ATTACHMENT G
COMPILER OPTION INFORMATION

"**PKG370**" is the command to invoke the compiler. The general format is:

```
PGK370  dsname {options}

    options:
                CLEAN
                ERROR(LIST)
                LIST(ERRI)
                RUN(TEXT)
```

PKG370 COMMAND

The PKG370 command is used to compile more than one Ada source file in a single compilation session. The PKG370 command accepts either an Ada program file or a file which contains a filelist of files containing compilation units.

Dsname specifies the file to be compiled. If ftype is not FILELIST, fname, ftype, and fmode is considered to be an Ada source file.

The **CLEAN** option is used to erase all files derived from a compilation after a main program has completed compilation and execution. These files include object files, listing files, error files, and execution produced files.

The **ERROR(LIST)** option creates a listing file only when errors are encountered. The file contains compile-time error messages interspersed with the source code.

The **LIST(ERRI)** option produces a compilation source listing. Semantic errors, syntax errors, and warnings are interspersed.

The **RUN(TEXT)** option causes the program to load and execute. It is assumed that the program displays the results on the console. The output of the entire compilation and execution is copied to a dataset. This dataset is examined to determine whether the program was executed successfully. The possible results are:

PASSED==	Indicates the program passed
FAILED**	Indicates the program failed
NOT-APPLICABLE++	Implies the program passed

This is compatible with the error reporting used by ACVC tests.

APPENDIX C

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in Chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this Appendix are to compiler documentation and not to this report. Implementation-specific portions of the package STANDARD, which are not a part of Appendix F, are:

```
package STANDARD is

   ...
   type INTEGER is range -2_147_483_648 .. 2_147_483_647;
   type SHORT_INTEGER is range -32_768 .. 32_767;

   type FLOAT is digits 6 range -7.23701E+75 .. 7.23701E+75;
   type LONG_FLOAT is digits 15 range -7.23700557733225E+75
                                   .. 7.23700557733225E+75;

   type DURATION is delta 2#1.0#E-14 range -86400.0 .. 86400.0;
   ...

end STANDARD;
```

ATTACHMENT A

APPENDIX F OF THE LANGUAGE REFERENCE MANUAL

The Ada language definition allows for certain target dependencies in a controlled manner. This section, called Appendix F as prescribed in the LRM, describes implementation-dependent characteristics of the IBM Ada/370, Version 1.1.0 running under CMS or MVS.

1. Implementation-Defined Pragmas

PRAGMA INTERFACE(Assembly, <subroutine_name>);

PRAGMA INTERFACE(Assembler, <subroutine_name>);

PRAGMA INTERFACE(Fortran, <subroutine_name>);

PRAGMA SUPPRESS_ALL;

to cause Pragma SUPPRESS to be invoked simultaneously for all the following condition_names: access_check, discriminant_check, index_check, length_check, division_check, elaboration_check, and storage_check.

PRAGMA NO_SUPPRESS (<identifier>);

to prevent the suppression of checks within a particular scope. Particularly useful when a section of code that relies upon predefined checks executes correctly, but, for performance reasons, the suppression of checks in the rest of the code is needed.

PRAGMA COMMENT (string_literal);

embeds string_literal into object code.

PRAGMA IMAGES (enumeration_type, <immediate>|<deferred>);

generates a table of images for the enumeration type. **deferred** causes the table to be generated only if the enumeration type is used in a compilation unit.

```
PRAGMA INTERFACE_INFORMATION
                (<name>,
                 <link_name>,
                 <mechanism>,
                 <parameters>,
                 <clobbered_regs>);
```

when used in association with pragma INTERFACE, will provide access to any routine whose name can be specified by an Ada string literal.

PRAGMA PRESERVE_LAYOUT (ON => <Record_Type_Name>);

forces the compiler to maintain the Ada source order of components of a given record type, thereby preventing the compiler from performing this record layout optimization.

*PRAGMA OS_TASK (priority);

to specify the relative urgency of each MVS task created.

*PRAGMA ALLOCATION_DATA

```
(<access_type>,
 <residence_mode>,
 <allocation_duration>,
 <subpool_number>,
 <discrete_user_data>);
```

to associate MVS virtual storage attributes with an Ada access type.

Note that PRAGMA OS_TASK and PRAGMA ALLOCATION_DATA are effective only when compiling for an MVS target. Both pragmas require that an MVS runtime be present.

2. Implementation-Defined Attributes

2.1. Integer Type Attributes

Extended_Image (Item, <Width>, <Base>, <Based>, <Space_IF_Positive>);

to return the image associated with Item as defined in Text_IO.Integer_IO. The Text_IO definition states that the value of Item is an integer literal with no underlines, no exponent, no leading zeroes (but a single zero for the zero value), and a minus sign if negative.

Extended_Value (Item);

to return the value associated with Item as defined in Text_IO.Integer_IO. The Text_IO definition states that given a string, it reads an integer value from the beginning of the string. The value returned corresponds to the sequence input.

Extended_Width (<Base>, <Based>, <Space_IF_Positive>);

to return the width for a subtype specified.

2.2. Enumeration Type Attributes

Extended_Image (Item, <Width>, <Uppercase>);

to return the image associated with Item as defined in Text_IO.Enumeration_IO. The Text_IO definition states that given an enumeration literal, it will output the value of the enumeration literal (either an identifier or a character literal). The character case parameter is ignored for character literals.

Extended_Value (Item);

to return the image associated with Item as defined in Text_IO.Enumeration_IO. The Text_IO definition states that it reads an enumeration value from the beginning of the given string and returns the value of the enumeration literal that corresponds to the sequence input.

Extended_Width;

to return the width for a specified subtype.

2.3. Floating Point Attributes

Extended_Image (Item, <Fore>, <Aft>, <Exp>, <Base>, <Based>);

to return the image associated with Item as defined in Text_IO.Float_IO. The Text_IO definition states that it outputs the value of the parameter Item as a decimal literal with the format defined by the other parameters. If the value is negative, a minus sign is included in the integer part of the value of Item. If Exp is 0, the integer part of the output has as many digits as are needed to represent the integer part of the value of Item or is zero if the value of Item has no integer part.

Extended_Value (Item);

to return the value associated with Item as defined in Text_IO.Float_IO. The Text_IO definition states that it skips any leading zeroes, then reads a plus or minus sign if present, then reads the string according to the syntax of a real literal. The return value is that which corresponds to the sequence input.

Extended_Digits (<Base>);

to return the number of digits using base in the mantissa of model numbers of the specified subtype.

2.4. Fixed Point Attributes

Extended_Image (Item, <Fore>, <Aft>, <Exp>, <Base>, <Based>);

to return the image associated with Item as defined in Text_IO.Fixed_IO. The Text_IO definition states that it outputs the value of the parameter Item as a decimal literal with the format defined by the other parameters. If the value is negative, a minus sign is included in the integer part of the value of Item. If Exp is 0, the integer part of the output has as many digits as are needed to represent the integer part of the value of Item or is zero if the value of Item has no integer part.

Extended_Value (Image);

to return the value associated with Item as defined in Text_IO.Fixed_IO. The Text_IO definition states that it skips any leading zeroes, reads a plus or minus sign if present, then reads the string according to the syntax of a real literal. The return value is that which corresponds to the sequence input.

Extended_Fore (<Base>, <Based>);

to return the minimum number of characters required for the integer part of the based representation specified.

Extended_Aft (<Base>, <Based>);

to return the minimum number of characters required for the fractional part of the based representation specified.

3. Package SYSTEM

The current specification of package SYSTEM is provided below.

```
With Unchecked_Conversion;

PACKAGE System IS

  --=====================================================
  -- CUSTOMIZABLE VALUES
  --=====================================================

  TYPE Name    IS (MC68000, ANUYK44, IBM370);
```

```
System_Name  : CONSTANT name := IBM370;

Memory_Size  : CONSTANT := (2 ** 24)-1;
Tick         : CONSTANT := 1.0 / (10 ** 6);

--==========================================================
-- NON-CUSTOMIZABLE, IMPLEMENTATION-DEPENDENT VALUES
--==========================================================

Storage_Unit : CONSTANT := 8;
Min_Int      : CONSTANT := -(2 ** 31);
Max_Int      : CONSTANT := (2 ** 31) - 1;
Max_Digits   : CONSTANT := 15;
Max_Mantissa : CONSTANT := 31;
Fine_Delta   : CONSTANT := 1.0 / (2 ** Max_Mantissa );

Subtype Priority IS Integer RANGE 0 .. 255;

--==========================================
-- ADDRESS TYPE SUPPORT
--==========================================

type Memory is private;
type Address is access Memory;

Null_Address : Constant Address := null;

type Address_Value is RANGE -(2**31) .. (2**31)-1;

Hex_80000000 : constant Address_Value := - 16#80000000#;
Hex_90000000 : constant Address_Value := - 16#70000000#;
Hex_A0000000 : constant Address_Value := - 16#60000000#;
Hex_B0000000 : constant Address_Value := - 16#50000000#;
Hex_C0000000 : constant Address_Value := - 16#40000000#;
Hex_D0000000 : constant Address_Value := - 16#30000000#;
Hex_E0000000 : constant Address_Value := - 16#20000000#;
Hex_F0000000 : constant Address_Value := - 16#10000000#;

function Location is new Unchecked_Conversion (Address_Value, Address);

function Label (Name: String) return Address;
pragma Interface (META, Label);

--===============
-- CALL SUPPORT
--==============

type Subprogram_Value IS
```

```
 record
   Proc_addr       : Address;
   Parent_frame    : Address;
 end record;

 Max_Object_Size    : CONSTANT := Max_Int;
 Max_Record_Count   : CONSTANT := Max_Int;
 Max_Text_Io_Count  : CONSTANT := Max_Int-1;
 Max_Text_Io_Field  : CONSTANT := 1000;

private
 type Memory is
 record
   null;
 end record;

end SYSTEM;
```

4. Representation Clauses

This implementation supports address, length, enumeration, and record representation clauses with the following exceptions:

> Address clauses are not supported for package, for entry, for tasktype, for subprograms.
>
> Enumeration clauses are not supported for boolean representation clauses.

The size in bits of representation specified records is rounded up to the next highest multiple of 8, meaning that the object of a representation specified record with 25 bits will actually occupy 32 bits.

Non-supported clauses are rejected at compile time.

5. Implementation-Generated Names

There are no implementation-generated names denoting implementation-dependent components. Names generated by the compiler shall not interfere with programmer-defined names.

6. Address Clause Expression Interpretation

Expressions that appear in Address clauses are interpreted as virtual memory addresses.

7. Unchecked Conversion Restrictions

Unchecked_Conversion is allowed except when the target data subtype is an unconstrained array or record type. If the size of the source and target are static and equal, the compiler will perform a bitwise copy of data from the source object to the target object.

Where the sizes of source and target differ, the following rules will apply:

- If the size of the source is greater than the size of the target, the high address bits will be truncated in the conversion.

- If the size of the source is less than the size of the target, the source will be moved into the low address bits of the target.

The compiler will issue a warning when Unchecked_Conversion is instantiated with unequal sizes for source and target subtype. Unchecked_Conversion between objects of different or non-static sizes will usually produce less efficient code and should be avoided, if possible.

8. Implementation-Dependent Characteristics of the I/O Packages

- Sequential_IO, Direct_IO, and Text_IO are supported.
- Low_Level_IO is not supported.
- Unconstrained array types and unconstrained types with discriminants may not be instantiated for I/O.
- File names follow the conventions and restrictions of the target operating system.
- In Text_IO, the type Field is defined as follows: subtype Field is integer range 0..1000;
- In Text_IO, the type Count is defined as follows: type Count is range 0..2_147_483_646;